SF 298 MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

# REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | 1996 | Technical |

4. TITLE AND SUBTITLE
Software maintenance and software Reuse

5. FUNDING NUMBERS
DAAH04-95-1-0250

6. AUTHOR(S)
Y. B. Reddy and Dachelle Weems

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)
Grambling State University
Department of Math and Computer Science
Grambling, LA 71245

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
ARO 34157.39-MA-ISP

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12 b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Maintenance is the process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adopt to a changed environment. Software maintenance has been classified into three categories: corrective, adaptive, and perfective. During the maintenance it may be necessary to disassemble an old system into components and resemble them into new system (re-engineering) with old and need new software components. In this paper we discuss the reuse of software components while reconstructing the system.

14. SUBJECT TERMS
Software reuse, Re-engineering, Software maintenance, reconstructive maintenance

15. NUMBER IF PAGES 14

16. PRICE CODE

| 17. SECURITY CLASSIFICATION OR REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UL |

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Enclosure 1

Y. B. Reddy and Dachelle Weems, Grambling State University, Grambling, LA 71245 - Software maintenance and software Reuse:-

Maintenance is the process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adopt to a changed environment. Software maintenance has been classified into three categories: corrective, adaptive, and perfective. During the maintenance it may be necessary to disassemble an old system into components and resemble them into new system (re-engineering) with old and need new software components. In this paper we discuss the reuse of software components while reconstructing the system.

Note: This research is supported by Advanced Distributed Simulation Research Consortium and Office of Naval Research

70th Annual Meeting of the Louisiana Academy of Sciences
Nicholls State University, Thibodeaux, LA, February 1-2, 1996.

The software maintenance has been classified into three categories:

**Corrective** - Performed to correct a discovered problem in a SW system

**Adaptive** - When software system has to adapt a new operational environment

**Perfective** - Make an existing software system perform better

Software systems are constructed from scratch. Many times a system will be constructed from reusable constants rather than constructed from scratch. for example:

> a set of new software requirements are often initiated by new product ideas that come directly from existing products.

Such constructions of new software systems are common in embedded software environment. They are constructed using existing set of requirements, design and implementation. This distinct type of software development can neither fit into a traditional software development framework, nor can it be classified by the known maintenance categories.

Since the reconstruction is not same as develop a system from scratch, the maintenance depends with *new software requirements, design, and implementation from existing systems*.

The properties of perfective and adaptive maintenance are closely related and useful but the constructive maintenance is different in many ways. We discuss this point in this paper.

## Reconstructive maintenance

The reconstructive maintenance is defined as the maintenance performed to accommodate some dramatic changes in both software requirements and hardware environment in existing systems. This kind of maintenance is quite common in the embedded software industries where new products are frequently introduced. The new products may be

- previously tested products in another system
- newly designed and implemented
- modified old products as required by new system requirements

In the case of modified old products, we may need to preserve certain functionalities.

Adaptive maintenance deals with maintenance performed to preserve the same functional requirements, whereas reconstructive maintenance deals with changes which include operational, functional, and environmental. The reconstructive maintenance has to consider *reusing other software components* when constructing a new one which does not exist in adaptive maintenance.

Reverse engineering deals with understanding of existing system and design the system to meet the new goals. The reconstructive maintenance moves one step further., that is to use reusable components and construct the system to meet new operational environment, hardware, software facilities.

Reconstructive maintenance do not correct any bug in the software and it does not perfect the existing system. These two characteristics are different from *reverse engineering*.

Reconstructive maintenance is to adapt new HW environment (not new development)
Reconstructive maintenance leads to a new system (it does not adapt old system)

The reconstructive maintenance disassembles the existing software into functionally independent modules which might be reused in a new system.
Reconstructive maintenance requires new modules in addition to the reused ones.

The reconstructive maintenance engineer keeps in mind the following points:

- understand new software requirements and old system
- separation of modules from old system and possible reuse in new system
- The construction of new modules in addition to reused ones

## We now discuss this problem in three important steps:

1. Understanding the application domain
2. Disassembling the existing system
3. Construction of new system

### Understanding the application domain:

Software engineer must have sufficient amount of knowledge of application domain (That is software engineer must be knowledgeable in application domain).

If software engineer has insufficient knowledge one should gain knowledge through class room or independent study.

### Disassembling the existing system:

After acquiring the domain knowledge, decompose the system into functional modules.

Form the reconstructive module set with appropriate requirements

identify reusable components from existing system, and other reusable components.

### Construction of new system:

Reconstructive system may be done in spiral model or with object-oriented concepts.
Analyze possible reuse of all component.
Use incremental building of new system
Test the new system

Example:

In this example we discuss constructive maintenance of a neural network model. We will discuss here the backpropagation model.

Backpropagation is a widely used neural network model during recent years for most of the pattern recognition problems [5]. Multilayer Backpropagation paradigm is a feedforward neural network having more than one hidden layer. In the present problem, the program written for backpropagation in C-language is selected to reverse engineering case study [6]. In the present research, the neural network program is used to identify a fault component in a hierarchically connected sensor output using the two gates "and" and "or" [7]. The 'and' gate takes the inputs and outputs the minimum value. The 'or' gate takes the inputs and outputs the maximum value. The syntax of the diagrams are shown in Figure 2. The network with one hidden layer with six inputs and seven outputs is given in Figure 1. The experiment was conducted with two hidden layers and projected test outputs. The main idea of conducting an experiment is to cover most of the lines of code and branches with test input values. Test coverage reports are also generated to find the behavior of the program. It is clear that the program [6] is well written but it misleads the user while executing the program particularly when calling the functions *'dread'* and *'dwrite'* in 'output_generation' (user needs to remember the previous data file name without extension after the period). The two modules adds the extensions as: dread adds _v to data file and (b) wtread adds _w to the data file to separate these from others in the directory. The program can be used to train various data files and generate outputs for any trained pattern. The execution of the program and the necessary modifications are discussed below. The program never keeps track of previous weights if a user wants to train with more than one input file or further trains the system with the same data again if the system does not reach the minimum required error. The design extraction using Ensemble documentation is shown in Figure 3. Each time the program executes as if it was started for the first time (weights are initialized each time). With little modifications the program can be further trained for more than one data file at different times so that we can save previous trained time. The design modifications with added functions are shown in Figure 3 (with dotted lines) and metric reports in Appendix A.

**References**


1. Chikofsky, E. J. and J. H. Cross III; Reverse engineering and design recovery: A taxonomy, IEEE software, Jan. 1990, p 13 - 17.
2. CASE Tools for Reverse Engineering. CASE outlook, Vol. 2, no. 2, 1988, p1-15.
3. T.A.Corbi; Program understanding: Challenge for the 1990s IBM Systems Journal, Vol. 28, No. 2, 1989, p 294-306.
4. Oman, P; Maintenance Tools, IEEE software, May 1990, p 21-23.
5. Rumelhart, D.E., Hinton, G.E., and Williams, R.J. Learning internal representations by error propagation Parallel Distributed processing, Vol. I, p 318-364, MIT press, 1987.
6. You-Han Pao; Adaptive Pattern Recognition and Neural Networks Addition-Wesley Publishing company, Inc. (1989)
7. Piotr Gmytrasiewicz, Jere A. Hassberger, and John C. Lee Fault Tree Based Diagnostics Using Fuzzy Logic; IEEE Trans. On Pattern Analysis and Machine Intelligence; Vol. 12, No. 11, Nov. 1990, p 1115-1119.
8. Ted J. Biggerstaff; Design recovery for maintenance and reuse, IEEE Computer, July 1989, p 36-49.
9. Cadre Technologies Inc.; 222 Richmond Street, Providence, RI 02903; Ph - 1-800-548-7645.
10. Robert S. Arnold (editor) Software Re-engineering IEEE Computer Society Press (1993)
11. Walter, Richard C., and Elliot Chikofsy Reverse Engineering progress Along Many Dimensions CACM No.5, Vol. 37, May 1994. pp 22-24
12. Clapp, Judith Designing Software for Maintainability Computer Design, September 1981. pp 197-204.

**Re-engineering to an Object-Oriented architecture**

**Object-oriented Development**

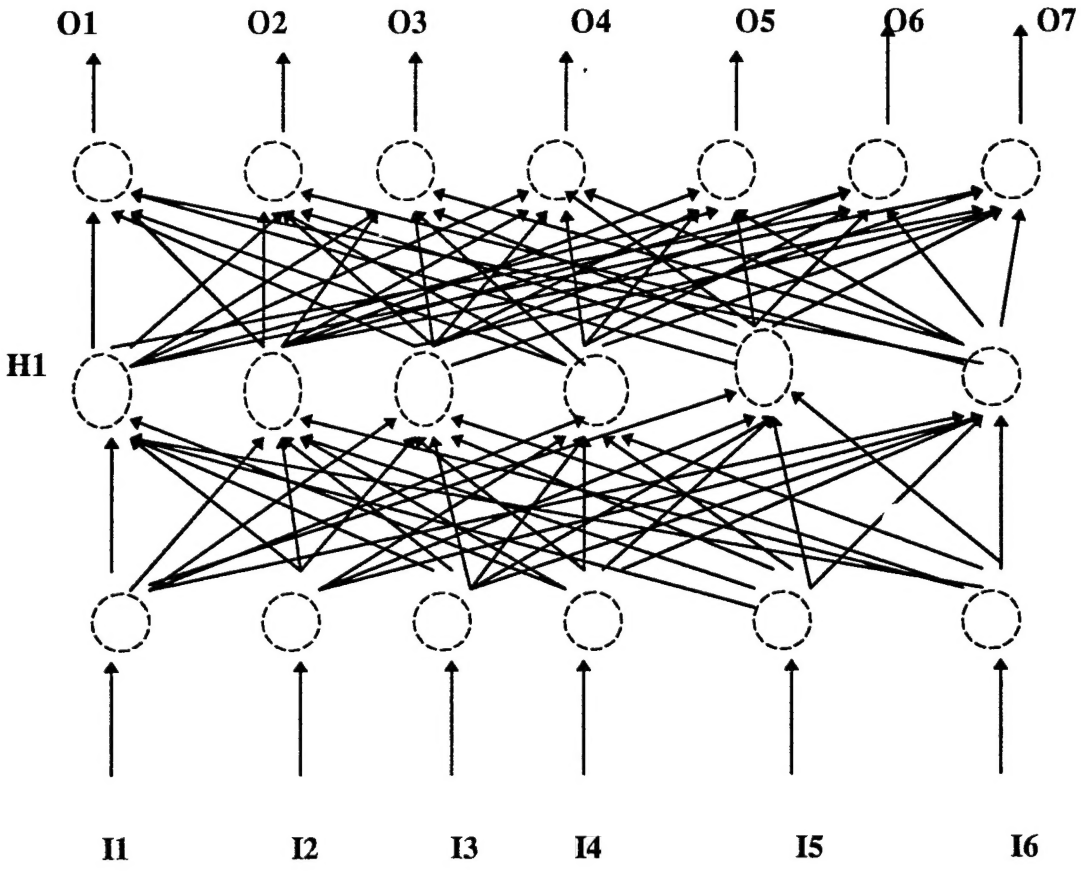


**Figure 1** **Neural Network model for 6 inputs and 7 outputs**

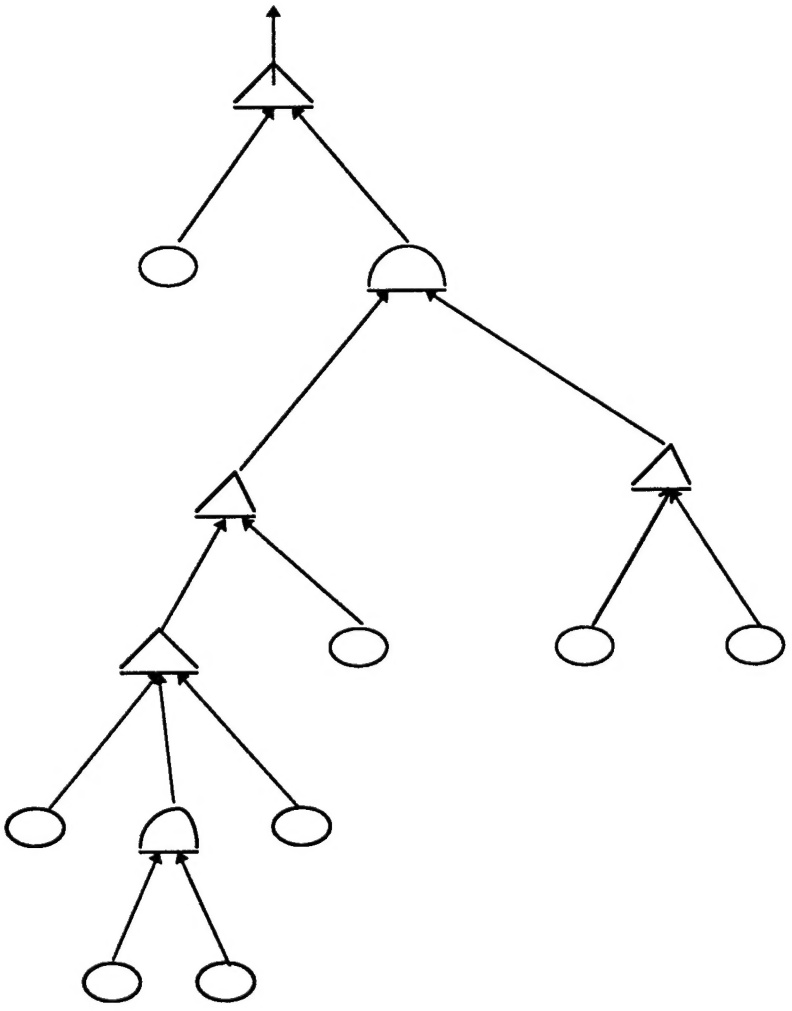

**Figure 2:** **Hierarchical Connection of sensors using 'and' , 'or' gates**

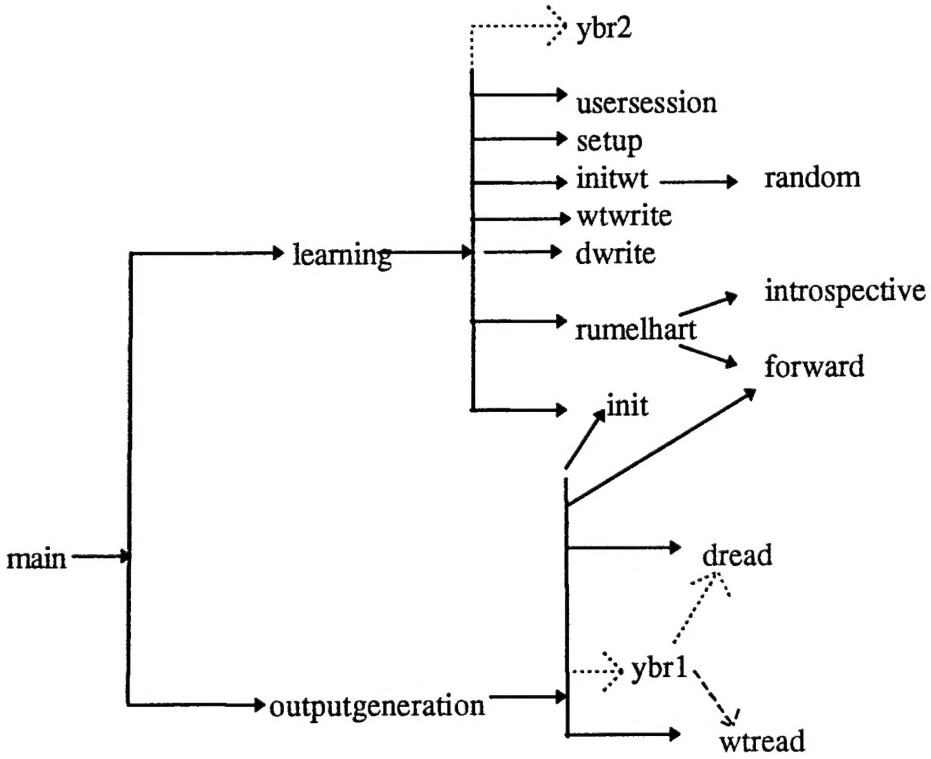


Figure 3: Design extraction using Ensemble program
(--> shows the design modification)

APPENDIX A

Function summary Report

---

Ensemble Report:
Metrics

---

This report contains metrics information about function in the model.
Model='bj1_user'

| | run1 | run2 | run3 |
|---|---|---|---|
| Average cyclometric complexity = | 6.13 | 6.0 | 5.47 |
| Average data complexity = | 6.30 | 6.3 | 5.91 |
| Number of total lines = | 396 | 390 | 405 |
| Number of lines executed = | 316 | 347 | 362 |
| Percent lines executed = | 79.80% | 88.97 | 89.38 |
| Number of total branches = | 151 | 147 | 149 |
| Number of branches executed = | 114 | 127 | 123 |
| Percent branches executed = | 75.50% | 86.39% | 82.55% |

**WARNING**: Metrics that could not be calculated for a function show up with ** for their metric value. For data complexity, check to make sure the function is part of your model. For coverage metrics, check to make sure the function is part of your coverage set. For branch coverage, this could mean there were no branches in the function.

Function Summary Report

| file | function | cyclo | | | data | | | %line | | | %branch | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | run1 | *run2* | **run3** | run1 | *run2* | **run3** | run1 | *run2* | **run3** | run1 | *run2* | **run3** |
| bj.c | dread | 4 | 4 | 4 | 11.0 | 11.0 | 11.0 | 0 | 70 | 70 | 0 | 66 | 66 |
| bj.c | dwrite | 6 | 6 | 6 | 14.0 | 14.0 | 14.0 | 80 | 80 | 80 | 80 | 80 | 80 |
| bj.c | forward | 6 | 6 | 6 | 8.0 | 8.0 | 8.0 | 100 | 100 | 100 | 100 | 100 | 100 |
| bj.c | init | 5 | 5 | 5 | 7.0 | 7.0 | 7.0 | 100 | 100 | 100 | 100 | 100 | 100 |
| bj.c | initwt | 3 | 3 | 3 | 2.50 | 2.5 | 2.5 | 100 | 100 | 100 | 100 | 100 | 100 |
| bj.c | introspective | 6 | 6 | 6 | 10.0 | 10.0 | 10.0 | 92 | 92 | 92 | 70 | 80 | 70 |
| bj.c | learning | 3 | 3 | 4 | 0.38 | 0.38 | 0.44 | 72 | 100 | 100 | 50 | 75 | 66 |
| bj.c | main | 7 | 7 | 7 | 0.67 | 0.67 | 1.0 | 82 | 82 | 82 | 55 | 55 | 55 |
| bj.c | output_generation 10 | 8 | 8 | 1.60 | 1.60 | 2.67 | 73 | 87 | 84 | 83 | 85 | 85 | |
| bj.c | random | 1 | 1 | 1 | 2.0 | 2.0 | 2.0 | 100 | 100 | 100 | ** | ** | ** |
| bj.c | rumelhart | 20 | 20 | 20 | 7.33 | 7.33 | 7.33 | 94 | 94 | 94 | 92 | 97 | 92 |
| bj.c | set_up | 2 | 2 | 2 | 11.00 | 11.0 | 11.0 | 100 | 100 | 100 | 100 | 100 | 100 |
| bj.c | user_session | 8 | 8 | 8 | 7.0 | 7.0 | 7.0 | 80 | 80 | 80 | 71 | 85 | 71 |
| bj.c | wtread | 5 | 5 | 5 | 6.0 | 6.0 | 6.0 | 0 | 72 | 72 | 0 | 75 | 75 |
| bj.c | wtwrite | 6 | 6 | 6 | 6.0 | 6.0 | 6.0 | 78 | 78 | 78 | 80 | 80 | 80 |
| bj.c | ybr1 | | | **1** | | | **0.5** | | | **100** | | | ** |
| bj.c | ybr2 | | | **1** | | | **4.0** | | | **100** | | | ** |

## Modified Code

```
output_generation( )
{
  int  i,j,m,nsample;
  char  ans[10];
  char  dfile[20];
/* If task is already in the memory, data files for task do not need to be
   read in. But, if it is a new task, data files should be read in to reconstruct the net */
   printf("\nGeneration of outputs for a new pattern");
   printf("\n\t Present task name is %s", task_name);
   printf("\n\t Work on a different task?   ");
   printf("\n\t   Answer yes or no  : ");
   scanf("%s", ans);
   if ((ans[0]=='y') || (ans[0]=='Y'))
   {
     printf("\n\t Type the task name : ");
     scanf("%s", task_name);
     dread(task_name);
     init();
     wtread(task_name);
   }
          /* input data for output generation are created */
   printf("\nEnter file name for patterns to be processed: ");
   scanf("%s",dfile);
   if ((fp1=fopen(dfile,"r")) == NULL)
   {
     perror("Cannot open dfile");
     exit(0);
   }
   printf("\nEnter number of patterns for processing: ");
   scanf("%d", &nsample);
   for (i=0; i<nsample; i++)
     for (m=0; m<ninattr;m++)
       fscanf(fp1,"%f",&input[i][m]);
          /* output generation calculation starts */
   for (i=0; i<nsample; i++)
   {
     forward(i);
     for (m=0; m<noutattr; m++)
       printf("\n sample %d output %d = %f",i,m,*(outptr[nhlayer+1]+m));
         printf("\n");
    }
   printf("\nOutputs have been generated ");
   if((i=fclose(fp1)) != 0)
     printf("\nFile cannot be closed %d",i);
}
```

## Appendix E

```
ybr1()
{
printf("\nIf you did not train the system and you want to use pre-trained
system\n");
printf("enter the data file name used previously for training without
extension\n");
printf("\nif you just train the system and use, please enter the same name\n");
printf("\nIf you do not follow the instructions program terminates or you get
bad results\n");
printf("\n\tType the task name:");
scanf("%s", task_name);
dread(task_name);
init();
wtread(task_name);
}
/*=========================================================*/
ybr2()
{
 printf("\nTo further train the system with another data file with same\n");
 printf(" number of inputs, outputs then enter the data file name\n");
 printf(" \nenter the name of the data file:");
 scanf("%s", task_name);
 printf("\nEnter total number of input samples in this data file:");
 scanf("%d", &ninput);
 printf("\nMax number of iterations?: ");
 scanf ("%d", &cnt_num);
printf("\nexecution starts....");
}
/*======== main body of learning=========      */
learning( )
{
  int result;
 if (In == 0)  {  user_session();     set_up();     init();  } else ybr2();
  do  {
    initwt();
    result = rumelhart(0,ninput);
  } while (result == RESTRT);
  if (result == FEXIT)     {
    printf("\n Max number of iterations reached,");
    printf("\n    but failed to decrease system");
    printf("\n error sufficiently");
  }
  dwrite(task_name);
  wtwrite(task_name);
}
```

```
        /* main body of output generation */
output_generation( )
{
  int i,j,m,nsample;
  char ans[10];
  char dfile[20];

/* If task is already in the memory, data files for task do not need to be read in.
  But, if it is a new task, data files should be read in to reconstruct the net     */
   printf("\nGeneration of outputs for a new pattern");
   printf("\n\t Present task name is %s", task_name);
   printf("\n\t Work on a different task?  ");
   printf("\n\t  Answer yes or no : ");
   scanf("%s", ans);
   if ((ans[0]=='y') || (ans[0]=='Y'))  ybr1();
/*
     {
        printf("\n\t Type the task name : ");
        scanf("%s", task_name);
        dread(task_name);
        init();
        wtread(task_name);
     }
*/
        /* input data for output generation are created */
   printf("\nEnter test data file name for patterns to be processed: ");
   scanf("%s",dfile);
   if ((fp1=fopen(dfile,"r")) == NULL)
   {
     perror("Cannot open dfile");
     exit(0);
   }
   printf("\nEnter number of patterns for processing: ");
   scanf("%d", &nsample);
   for (i=0; i<nsample; i++)
     for (m=0; m<ninattr;m++)
       fscanf(fp1,"%f",&input[i][m]);
        /* output generation calculation starts */
   for (i=0; i<nsample; i++)
   {
     forward(i);
     for (m=0; m<noutattr; m++)
       printf("\n sample %d output %d = %f",i,m,*(outptr[nhlayer+1]+m));
        printf("\n");
    }
   printf("\nOutputs have been generated ");
   if((i=fclose(fp1)) != 0)
     printf("\nFile cannot be closed %d",i);
}
```

```
/************************ MAIN ***************************/
main()
{
  char select[20], cont[10];
   char yb[5];
  strcpy(task_name, "*********");
printf("you want to use prelearned system or train first time: enter p or f: ");
scanf("%s",yb);
if (yb[0]=='f') { ln = 0;   printf("\nselect learning\n");
       else { ln = 1; printf("\nselect output generation \n"); }

  do {
    printf("\n** Select L(earning) or O(utput generation) **\n");
    do {
      scanf ( "%s",select);
      switch(select[0]) {
        case 'o':
        case 'O':
          output_generation ();
          break;
        case 'l':
        case 'L':
          learning();
          break;
        default:
          printf("\nanswer learning or output generation ");
          break;
       }
    } while (( select[0]!='o') && (select[0]!='O')
        && (select[0]!='l') && (select[0]!='L'));
    printf("\nDo you want to continue?  ");
    scanf("%s",cont);  ln += 1;
  }  while ((cont[0] =='y') || (cont[0] =='Y'));

  printf("\nIt is all finished. ");
  printf("\n Good bye  ");
}
```